Linguaggi di Programmazione e Compilatori
A.A. 2009/2010
Homeworks – Analisi semantica e JavaCC
Doc. Maria Rita Di Berardini

Questa raccolta di esercizi sostituisce l'ultima parte dello scritto (ma **non** dell'orale) dell'esame di Linguaggi di Programmazione e Compilatori. È la terza serie di "homeworks" previsti dal docente durante il periodo didattico (tre in totale).

Regolamento

- per poter accedere all'orale è necessario *sostenere* e *superare* tutti gli homeworks assegnati.
- La soluzione degli esercizi proposti deve essere consegnata tassativamente entro le ore *18:30* di Martedì **15 giugno 2010**. La consegna può essere effettuata tramite posta elettronica (inviare una mail con oggetto *Analisi semantica* a mariarita.diberardini@unicam.it) o di persona (presso il mio ufficio, ultimo piano del polo didattico).
- Tutte le soluzioni consegnate oltre tempo massimo (ore 18:30 del **15 giugno 2010**) *non verrano prese in considerazione.*

Esercizi

1. Definire formalmente i seguenti concetti:
   (a) Definizioni guidate dalla sintassi (SDD)
   (b) Attributi sintetizzati
   (c) Attributi derivati
2. Si consideri la seguente SDD

PRODUZIONI	REGOLE SEMANTICHE
$E \to E_1 + T$	$\texttt{E.val} = \texttt{E}_1\texttt{.val} + \texttt{T.val}$
$E \to E_1 - T$	$\texttt{E.val} = \texttt{E}_1\texttt{.val} - \texttt{T.val}$
$E \to T$	$\texttt{E.val} = \texttt{T.val}$
$T \to T_1 * F$	$\texttt{T.val} = \texttt{T}_1\texttt{.val} * \texttt{F.val}$
$T \to T_1/F$	$\texttt{T.val} = \texttt{T}_1\texttt{.val}/\texttt{F.val}$
$T \to F$	$\texttt{T.val} = \texttt{F.val}$
$F \to (E)$	$\texttt{F.val} = \texttt{E.val}$
$F \to \textbf{num}$	$\texttt{F.val} = \textbf{num}\texttt{.lexval}$

   (a) Fornire l'albero di derivazione per la stringa $(13 + 4) * 3 + 11$.
   (b) Fornire l'albero di derivazione annotato per la stringa $(13 + 4) * 3 + 11$.

3. Data la seguente SDD

PRODUZIONI	REGOLE SEMANTICHE
$Decl \rightarrow TypeVarList$	`VarList.type` = `Type.type`
$Type \rightarrow$ **integer**	`Type.type` = `int`
$Type \rightarrow$ **float**	`Type.type` = `float`
$VarList \rightarrow$ **id**, $VarList_1$	$\texttt{VarList}_1\texttt{.type} = \texttt{VarList.type}$ **id**.`type` = `VarList.type`
$VarList \rightarrow$ **id**	**id**.`type` = `VarList.type`

   (a) Fornire l'albero di derivazione per la stringa **float id**, **id**, **id**.

   (b) Fornire il grafo delle dipendenze per la stringa **float id**, **id**, **id**.

   (c) Fornire un possibile ordinamento topologico del grafo delle dipendenze al punto (b).

4. Si considerino di nuovo le SDD degli esercizi ai punti 2 e 3. Dire se ognuna di esse è S-attributed, L-attributed, o nessuna delle precedenti.

5. Si consideri la seguente SDD

PRODUZIONI	REGOLE SEMANTICHE

$S \rightarrow A\,B$	`S.attr` = `A.attr`
	`A.attr` = `B.attr`
	`B.attr` = `S.attr` + `1`

$A \rightarrow a$	
$B \rightarrow b$	

   Riusciamo a valutare correttamente le regole semantiche associate alla produzione $S \rightarrow A\,B$? Motivare la risposta fornita.

6. La seguente grammatica libera da contesto genera tutte le stringhe costituite da una posizione iniziale – la coppia di coordinate cartesiane generata dal non terminale *Start* – e da una sequenza (eventualmente vuota) di spostamenti a sinistra, a destra, in basso oppure in alto.

$$
\begin{aligned}
&S \rightarrow Start\,Moves\\
&Start \rightarrow (\textbf{float}, \textbf{float})\\
&Moves \rightarrow \textbf{left}\,Moves\\
&Moves \rightarrow \textbf{right}\,Moves\\
&Moves \rightarrow \textbf{down}\,Moves\\
&Moves \rightarrow \textbf{up}\,Moves\\
&Moves \rightarrow \epsilon
\end{aligned}
$$

   Fornire una SDD che, per ogni stringa generata da tale grammatica, fornisca la posizione raggiunta a partire dalla posizione iniziale una volta eseguiti tutti gli spostamenti elencati nella sequenza. Fornire, inoltre, il grafo delle dipendenze, e relativo ordinamento topologico, per la stringa

(3.2, 1) **left up up right down left**

7. Si consideri la seguente specifica JavaCC

```
PARSER_BEGIN(Ex1)
  class Ex1{
    public static void main(String[] args)
      throws ParseException, TokenMgrError{
        Ex1 parser = new Ex1(System.in);
        int x = parser.S();
        System.out.println(x);
    }
  }
PARSER_END(Ex1)

int S():
{
  int x = 0;
}
{
    "a" x = S() "a" {return x + 2;}
  |
    "@" {return x;}
}
```

   (a) Fornire una grammatica libera da contesto in grado di generare tutte le stringhe parsate correttamente da tale specifica.

   (b) Quale è output prodotto nel caso in cui si compili una stringa sintatticamente corretta? Motivare la risposta fornita.

8. Fornire una specifica JavaCC in grado di implementare la SDD dell'esercizio al punto 6.

9. Quale tipo di 'warning' viene prodotto dalla compilazione della seguente specifica JavaCC?

```
PARSER_BEGIN(Ex2)
  class Ex2{
    public static void main(String[] args)
      throws ParseException, TokenMgrError{
        Ex2 parser = new Ex2(System.in);
        parser.S();
    }
  }
PARSER_END(Ex2)

void S(): {}
{
    "a"  S() "b"
  |
    "a"
}
```

   Fornire una nuova specifica in grado di risolvere il problema evidenziato da tale warning motivando opportunamente le scelte effettuate.

10. Riuscireste ad adottare la soluzione fornita per l'esercizio al punto 9 anche per la seguente specifica? Motivare opportunamente la risposta fornita.

```
PARSER_BEGIN(Ex3)
  class Ex3{
    public static void main(String[] args)
      throws ParseException, TokenMgrError{
        Ex3 parser = new Ex3(System.in);
        parser.S();
    }
  }
PARSER_END(Ex2)

void S(): {}
{
    "a"  S() "a"
  |
    "a"
}
```